Data & Knowledge Engineering 68 (2009) 665–682

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Schema exchange: Generic mappings for transforming data and metadata ☆

Paolo Papotti [1], Riccardo Torlone *

Università Roma Tre, Dipartimento di Informatica e Automazione, Via della Vasca Navale, 79 – 00146 Roma, Italy

ARTICLE INFO

Article history:
Available online 27 February 2009

Keywords:
Data exchange
Schema exchange
Schema templates
Schema mappings
Dependencies
Universal solutions

ABSTRACT

In this paper we present and study the problem of schema exchange, a natural extension of the data exchange problem in which mappings are defined over classes of similar schemas. To this end, we first introduce the notion of schema template, a tool for the representation of a set of schemas sharing the same structure. We then define the schema exchange notion as the problem of: (i) taking a schema that matches a source template, and (ii) generating a new schema for a target template, on the basis of a mapping between the two templates defined by means of FO dependencies. This framework allows the definition, once for all, of generic transformations that can be applied to different schemas. A method for the generation of a "correct" solution of the schema exchange problem is proposed and a number of general results are given. We also show how it is possible to generate automatically, from a schema exchange solution, a data exchange setting that reflects the semantics of the mappings between the original templates. This allows the definition of queries to migrate data from a source database into the one obtained as a result of a schema exchange.

1. Introduction

In the last years, we have witnessed an increasing complexity of database applications, especially when several, possibly autonomous data sources need to be accessed, transformed and combined. There is a consequent growing need for advanced tools and flexible techniques supporting the management, the exchange, and the integration of different and heterogeneous sources of information.

In this trend, the data exchange problem has received recently great attention, both from a theoretical [12,14] and a practical point of view [27]. In a data exchange scenario, given a mapping between a schema, called the source schema, and another schema, called the target schema, the goal is to take data structured under the source schema and transform it into a format conforming to the target schema.

In this paper, we address the novel problem of *schema* exchange, which naturally extends the data exchange process to *sets* of similar schemas: while the data exchange process operates over specific source and target schemas, the goal of schema exchange is rather the definition of generic transformations of data under structurally similar schemas. To this aim, we introduce the notion of *schema template*, which is used to represent a class of different database schemas sharing the same structure. Then, given a mapping between the components of a source and a target template, the goal is the translation of any database whose schema conforms to the source template into a format conforming to the target template.

☆ A preliminary version this paper appeared, under the title "Schema Exchange: A Template-Based Approach to Data and Metadata Translation", in the Proceedings of the *26th International Conference on Conceptual Modeling (ER), 2007*.

* Corresponding author. Tel.: +39 06 5733 3377; fax: +39 06 5733 3612.
E-mail addresses: papotti@dia.uniroma3.it (P. Papotti), torlone@dia.uniroma3.it (R. Torlone).

[1] Partially supported by an IBM Faculty Award.

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.02.005

This framework can be used to support several activities involved in the management of heterogeneous data sources. First, it allows the definition, once for all, of "generic" transformations that work for different but similar schemas, such as the denormalization of a pair of relation tables based on a foreign key between them. Then, it can support the *reuse* of a data exchange setting, since a mapping between templates can be derived from a mapping between schemas for later use in similar scenarios. Finally, it can be used to implement *model translations*, that is, translations of schemas and data from one data model to another (e.g., from relational to XML), a problem largely studied in recent years [1,4,24].

As has been done for data exchange [14], we introduce a formal notion of *solution* for the schema exchange problem and present a technique for the automatic generation of solutions. This is done by representing constraints over templates and mappings between them with a special class of first order formulas. These dependencies are used to generate the solution by applying the chase procedure [2] to an "encoding" of the source schema. Then, we propose an algorithm for deriving automatically a data exchange setting from a schema exchange solution and we show that this setting reflects the semantics of the mappings defined over templates at a higher level of abstraction. In this way, it is possible to migrate data from a source database into the database obtained as a result of the schema exchange.

The final goal of our research is the development of a design tool in which the user can: (i) describe data collections presenting structural similarities, by means of a source template T_1, (ii) define the structure of a possible transformation of the source by means of a target template T_2 and a set of correspondences over T_1 and T_2, graphically represented by lines between T_1 and T_2, (iii) translate any data source over a schema matching with T_1 into a format described by a schema matching with T_2, (iv) make use of a set of predefined schema exchange settings as *design patterns* for the development of new data exchange settings, and (v) convert a data exchange setting into a schema exchange one for its reuse.

The structure of the paper is as follows. In Section 2 we illustrate and motivate the schema exchange problem by means of a number of practical examples. In Section 3 we set the basic definitions and recall some useful results on the data exchange problem. In Section 4 we introduce formally the notion of template and show how templates and schemas are related. In Section 5, we define the problem of schema exchange and show how "correct" solutions can be generated and, in Section 6, we present a method for converting a schema exchange into a data exchange setting. Finally, in Section 7 we compare our work to existing literature and in Section 8 we draw some conclusions and sketch future directions of research.

2. Motivating examples

A graphical example of a data exchange scenario is reported in Fig. 1. On the left-hand side there is a *source* schema involving the relations `Emp` and `Dept` linked by a foreign key and, on the right-hand side, a *target* schema that involves the single relation `Employee`. The correspondences between the source and the target, expressed by means of arrows, suggest that the target can be obtained from the source by joining the relations `Emp` and `Dept` according to the foreign key, projecting the result on the attributes involved by the arrows, and storing the result into the relation `Employee`. This transformation can be specified, in a precise way, by means of the following first order rule, called *source-to-target dependency*:

$$\texttt{Emp}(e,n,s,d),\texttt{Dept}(d,dn,b) \rightarrow \texttt{Employee}(e,n,s,dn,b)$$

The goal of the given scenario is indeed to derive in an automatic way the queries needed to translate data structured under the source schema into a format conforming to the target schema, according to the given source-to-target dependencies.

Let us now consider the data exchange setting reported graphically in Fig. 2. The example is different (we now have students and courses) but the underlying transformation to obtain the target is indeed similar: again, we need to "denormalize" the source by joining two relations using a referential constraint between them.

The basic idea of our approach is that this transformation can be conveniently represented, in general terms, by means of a framework, which we have called *schema exchange*, reported graphically in Fig. 3. In this scenario we have *templates* of schema, that is, generic structures representing a set of database schemas with a similar configuration, and a mapping between them, again informally represented by means of arrows. The aim is to represent a generic transformation that can be applied to different, but structurally similar, data sources.

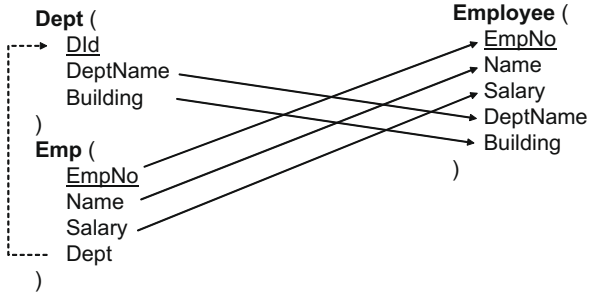

Fig. 1. An example of data exchange.

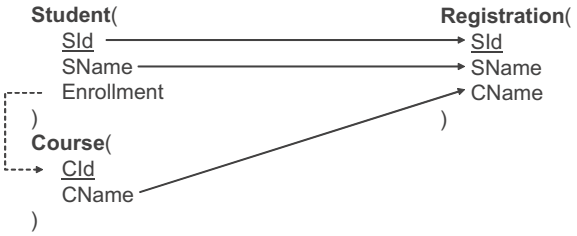

Fig. 2. Another example of data exchange.

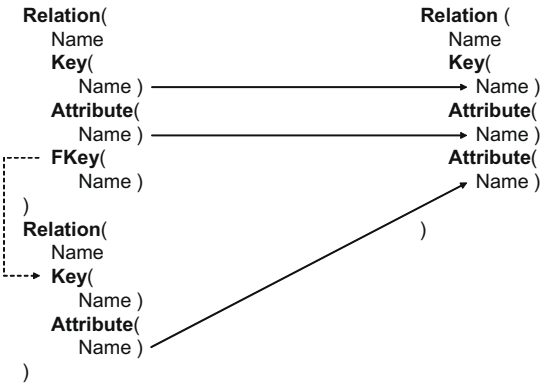

Fig. 3. An example of schema exchange that generalizes the data exchanges in Figs. 1 and 2.

Intuitively, the semantics of this scenario, in the example at hand, is the following. Given a relational schema involving at least two relations related by a foreign key, *for each* pair of relations R and R' such that: (i) R has a key, a non-empty set of non-key, non-foreign-key attributes and at least a foreign key attribute that refers to (the key of) R', and (ii) R' has a key and a non-empty set of non-key, non-foreign-key attributes, *generate* a target relation with the same attributes and the same key attributes of R, and with the attributes of R' not involved in the key.

Also in this case, the semantics can be precisely specified by a source-to-target dependency defined over templates, as follows:

$$\mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{Attribute}(n_A, n_R), \mathsf{FKey}(n_F, n_R, n'_R), \mathsf{Relation}(n'_R), \mathsf{Key}(n'_K, n'_R), \mathsf{Attribute}(n'_A, n'_R)$$
$$\rightarrow \mathsf{Relation}(n''_R), \mathsf{Key}(n_K, n''_R), \mathsf{Attribute}(n_A, n''_R), \mathsf{Attribute}(n'_A, n''_R)$$

We will make clear syntax and semantics of this rule later. Intuitively, it specifies that given a pair of (generic) relations in the source named n_R and n'_R, linked by a foreign key named n_F, there exists a relation in the target named n''_R with the same key as n_R and the attributes of both n_R and n'_R. Actually, we will show that this rule: (i) captures both the data exchange settings reported in Figs. 1 and 2, and (ii) can be used to generate the source-to-target dependencies reflecting the semantics of the mappings at the schema exchange level, described graphically in Fig. 3.

The notion of schema exchange can be used to support several practical problems related to the exchange of information between heterogeneous repositories of data:

- First, it can be used to support the design of a data exchange setting, since a schema exchange models naturally a *design pattern* for data exchange, that is, a general repeatable solution to a commonly occurring data transformation from a format into another;
- Moreover, it can support the *reuse* of a data exchange setting, since a schema exchange that generalizes a given data exchange mapping can be derived and later used to implement a transformation similar to the original one;
- Finally, a schema exchange framework can be used to represent and tackle the problem of *model translation* [1,4,24], in which the goal is the translation of schemas and data between heterogeneous data models. In fact, even if we mainly refer to the relational model, our notion of schema template can represent indeed schemas of a large variety of data models.

The rest of this paper is devoted to the precise definition of this notion of schema exchange, and the investigation of its general properties.

We point out that, as we have said in the introduction, the final goal of our research is the design of a practical tool, based on schema exchange, supporting the user in the design, reuse and maintenance of data exchange settings. The aim of the present study is providing a solid, theoretical basis for this tool.

3. Background

In this section we illustrate the notions that are at the basis of our work: schemas of the relational model, data dependencies and the problem of data exchange. We note that, in this paper, we make use of "database" terminology in which a *schema* is the description of the database structure and a (data) *model* provides a set of constructs for the specification of schemas (e.g., schemas are defined as relations in the relational model). In other contexts (for instance in OMG [22]) schemas are called *models*, and models are called *metamodels*.

3.1. Schemas and dependencies

A *schema* **S** is composed by a set of *relations* $R(A_1, \ldots, A_n)$, where R is the *name* of the relation and $A_1, \ldots, A_n$ are its *attributes*. Each attribute is associated with a set of values called the *domain* of the attribute. An *instance* of a relation $R(A_1, \ldots, A_n)$ is a finite set of tuples, each of which associates with each A_i a value taken from its domain. An instance I of a schema **S** contains an instance of each relation in **S**.

A *dependency* over a schema **S** is a first order formula of the form: $\forall \mathsf{x}(\phi(\mathsf{x}) \rightarrow \chi(\mathsf{x}))$ where $\phi(\mathsf{x})$ and $\chi(\mathsf{x})$ are formulas over **S**, and **x** are the free variables of the formula, ranging over the domains of the attributes occurring in **S**.

We will focus on two special kinds of dependencies: the tuple generating dependencies (tgd) and the equality generating dependencies (egd), as it is widely accepted that they include all of the naturally-occurring constraints on relational databases. A tgd has the form: $\forall \mathsf{x}(\phi(\mathsf{x})) \rightarrow \exists \mathsf{y}(\psi(\mathsf{x}, \mathsf{y}))$ where $\phi(\mathsf{x})$ and $\psi(\mathsf{x}, \mathsf{y})$ are conjunctions of atomic formulas. If **y** is empty (no existentially quantified variables), then we speak about a *full* tgd. An egd has the form: $\forall \mathsf{x}(\phi(\mathsf{x}) \rightarrow (x_1 = x_2))$ where $\phi(\mathsf{x})$ is a conjunction of atomic formulas and x_1, x_2 are variables in **x**. For simplicity, hereinafter we will omit all the quantifiers in formulas assuming that variables occurring in the left-hand size of a formula are universally quantified and those occurring only in the right-hand size are existentially quantified.

Example 1. Given a schema composed by the relations:

$$\mathsf{Department}(\mathsf{Did}, \mathsf{DeptName}), \mathsf{Employee}(\mathsf{Empno}, \mathsf{Name}, \mathsf{Dept})$$

a dependency of the form: $\mathsf{Department}(x_1, x_2), \mathsf{Department}(x_1, x_2') \rightarrow (x_2 = x_2')$ is an egd stating that Did is a key for Department, whereas a dependency of the form: $\mathsf{Employee}(x_1, x_2, x_3) \rightarrow \mathsf{Department}(x_3, x_4)$ is a tgd stating that there is a foreign key from the attribute Dept of Employee to the attribute Did of Department.

3.2. Data exchange

In the relational-to-relational data exchange framework [12], a *data exchange setting* is described by a four-tuple $M = (\mathsf{S}, \mathsf{T}, \Sigma_{st}, \Sigma_t)$, where:

- **S** is a source schema,
- **T** is a target schema,
- Σ_{st} is a finite set of *s–t* (source-to-target) tgds $\phi(\mathsf{x}) \rightarrow \chi(\mathsf{x}, \mathsf{y})$ where $\phi(\mathsf{x})$ is a conjunction of atomic formulas over **S** and $\chi(\mathsf{x}, \mathsf{y})$ is a conjunction of atomic formulas over **T**, and
- Σ_t is a finite set of tgds or egds over **T**.

Given an instance I of **S**, a solution for I under M is an instance J of **T** such that (I, J) satisfies Σ_{st} and J satisfies Σ_t. A solution may have distinct labelled nulls denoting unknown values.

Example 2. The transformation described graphically in Fig. 1 is precisely captured by the following data exchange setting:

- $\mathsf{S} = \{\mathsf{Emp}(\mathsf{EmpNo}, \mathsf{Name}, \mathsf{Salary}, \mathsf{Dept}),\ \mathsf{Dept}(\mathsf{DId}, \mathsf{DeptName}, \mathsf{Building})\}$.
- $\mathsf{T} = \{\mathsf{Employee}(\mathsf{EmpNo}, \mathsf{Name}, \mathsf{Salary}, \mathsf{DeptName}, \mathsf{Building})\}$.
- $\Sigma_{st} = \{\mathsf{Emp}(x_1, x_2, x_3, x_4), \mathsf{Dept}(x_4, x_5, x_6) \rightarrow \mathsf{Employee}(x_1, x_2, x_3, x_5, x_6)\}$.
- $\Sigma_t = \{\mathsf{Employee}(x_1, x_2, x_3, x_5, x_6), \mathsf{Employee}(x_1, x_2', x_3', x_5', x_6') \rightarrow (x_2 = x_2'), (x_3 = x_3'), (x_5 = x_5'), (x_6 = x_6')\}$.

Given the following instance I of the source schema **S**:

Emp

EmpNo	Name	Salary	Dept
234	John	50K	D2

Dept

DId	DeptName	Building
D2	Management	Harrison

A solution for I under the above data exchange setting is the following instance J of **T**:

Employee

EmpNo	Name	Salary	DeptName	Building
234	John	50K	Management	Harrison

Note that, according to the definition, other tuples can occur in the target, possibly involving values that do not occur in the source.

According to the final observation in the example above, there are in general many possible solutions for I under M. A solution J is *universal* if there is a homomorphism from J to every other solution for I under M. A homomorphism from an instance J to an instance J' is a function h from constant values and nulls occurring in J to constant values and nulls occurring in J' such that: (i) it is the identity on constants (that is, on the values occurring in the instance), and (ii) for each tuple $t = (a_1, \ldots, a_k)$ of J we have that $h(t) = (h(a_1), \ldots, h(a_k))$ is a tuple of J'.

In [14] it was shown that, when a solution for an instance I under M exists and the dependencies in Σ_t are either egds or *weakly acyclic* tgds (a class of tgds which admits limited forms of cycles among different relation arguments) a universal solution for I under M can be computed in polynomial time by applying the *chase procedure* to I using $\Sigma_{st} \cup \Sigma_t$. This procedure requires a preliminary notion: a *substitution* is a function s from constant values and variables to variables, constant values, and nulls that is the identity on constants. The chase takes as input an instance I and generates another instance by applying *chase steps* based on the given dependencies. There are two kinds of chase steps:

- A tgd $\phi(\mathrm{x}) \rightarrow \psi(\mathrm{x}, \mathrm{y})$ can be applied to I if there is a substitution s such that the application of s to variables and constants occurring in each atom of $\phi(\mathrm{x})$ produces a tuple in I; in this case, the result of its application is $I \cup s'(A)$, for each atom A occurring in $\psi(\mathrm{x}, \mathrm{y})$, where s' is the extension of s to **y** obtained by assigning fresh labelled nulls to the variables in **y**;
- An egd $\phi(\mathrm{x}) \rightarrow (x_1 = x_2)$ can be applied to I if there is a substitution s such that: (i) the application of s to variables and constants occurring in each atom of $\phi(\mathrm{x})$ produces a tuple in I and (ii) $s(x_1) \neq s(x_2)$; in this case, the result of its application is the following: if one of $s(x_1)$ and $s(x_2)$ is a constant and the other is a null, then the null is changed to the constant, otherwise the nulls are equated unless they are both constants, since in this case the process *fails*.

The chase of I is obtained by applying all applicable chase steps exhaustively to I.

Example 3. It is easy to see that the tgd in Σ_{st} of Example 2 can be applied to the instance I since there is a substitution that maps x_1 to `234`, x_2 to `John`, x_3 to `50K`, x_4 to `D2`, x_5 to `Management`, and x_6 to `Harrison`. The result of the application is the tuple in the target instance J. This is indeed the only applicable chase step.

Universal solutions are particularly important also for query answering, since Fagin et al. [14] have shown that the (certain) answers to a query q over the target schema can be obtained by evaluating q over any universal solution.

The problem is that, in general, there may exist multiple, different universal solutions for a data exchange problem. However, in [12] it has been shown that there is one particular universal solution, called the *core*, which is the most compact one. More specifically, the core of a universal solution J is (up to isomorphism) the smallest subset J^* of J such that J^* is homomorphically equivalent to J. Gottlob and Nash [16] have shown that the core of a universal solution of a data exchange problem, whose source-to-target constraints are tgds and whose target constraints consist of egds and weakly acyclic tgds, can be computed in polynomial time.

4. Schema templates and encoding of schemas

In this section, we introduce the notion of *schema template*, which is used to encode database schemas sharing the same structure. We also show how instances of templates, called e-schemas, can be converted to relational schemas and vice versa.

4.1. Schema templates

We fix a finite set $\mathscr{C}$ of *construct names*. A *construct* $C(p_1, \ldots, p_k)$ has a name C in $\mathscr{C}$ and a finite set $p_1, \ldots, p_k$ of distinct *properties*, each of which is associated with a set of values called the *domain* of the property. In principle, the set $\mathscr{C}$ can contain constructs from several data models so that we can include in our framework schemas of different models. In this paper however, for sake of simplicity, we focus on the relational model; the approach can be extended to richer data models, but challenging issues already arise in this setting.

Therefore, given a set $\mathscr{R}$ of *relation names*, we fix the following relational constructs and properties. For each construct, the properties that allow the identification of the construct are underlined.

Construct names	Properties (domain)
Relation	name ($\mathscr{R}$)
Attribute	name (string), nullable (boolean), in ($\mathscr{R}$)
Key	name (string), in ($\mathscr{R}$)
FKey	name (string), in ($\mathscr{R}$), refer ($\mathscr{R}$)

The Relation construct has only the name property, whose domain is $\mathscr{R}$. The constructs Key and FKey denote attributes that belong to a primary key and to a foreign key, respectively. The property in of the constructs Attribute, Key and FKey has $\mathscr{R}$ as domain and is used to specify the relation that includes the construct. Finally, the property refer of the construct FKey has also $\mathscr{R}$ as domain and is used to specify the relation it refers to. Clearly, other properties can be considered for every construct and have not been included to keep the notation simple. For instance, we could associate the properties basic_type and default_value with the construct Attribute.

Basically, a template is a set of constructs with a set of dependencies associated with them, which are used to specify constraints over single constructs and semantic associations between different constructs.

Definition 1. Template A (*schema*) *template* is a pair $(\mathsf{C}, \Sigma_{\mathsf{C}})$, where C is a finite collection of constructs and Σ_{C} is a set of dependencies over C.

Example 4. An example of a template $\mathscr{T} = (\mathsf{C}, \Sigma_{\mathsf{C}})$ contains the following set of constructs:

$$\mathsf{C} = \{\mathsf{Relation}(\mathsf{name}), \mathsf{Key}(\mathsf{name}, \mathsf{in}), \mathsf{Attribute}(\mathsf{name}, \mathsf{nullable}, \mathsf{in}), \mathsf{FKey}(\mathsf{name}, \mathsf{in}, \mathsf{refer})\}$$

and the dependencies:

$$\Sigma_{\mathsf{C}} = \{d_1 = \mathsf{Key}(n_K, n_R) \rightarrow \mathsf{Relation}(n_R), d_2 = \mathsf{Attribute}(n_A, u, n_R) \rightarrow \mathsf{Relation}(n_R), d_3 = \mathsf{FKey}(n_F, n_R, n'_R) \rightarrow \mathsf{Relation}(n_R), \mathsf{Relation}(n'_R), d_4 = \mathsf{Attribute}(n_A, u, n_R) \rightarrow (u = \mathtt{true})\}$$

The tgds d_1 and d_2 state the membership of keys and attributes to relations, respectively. The dependency d_3 states the membership of a foreign key to a relation and its reference to another relation. Finally, the egd d_4 states that we are considering only relations with attributes that allow null values.

Note that the dependencies d_1, d_2 and d_3 in Example 4 should always hold in a relational schema. Actually, they can be considered as axioms of the relational model. For this reason, in the following we will call them *relational* dependencies and assume that they always belong to the dependencies associated with a relational template.

Let us now introduce the notion of e-schemas. Basically, an e-schema corresponds to the *encoding* of a (relational) schema and is obtained by instantiating a template.

Definition 2. e-Schemas An *e-schema component S* over a construct *C* is a function that associates with each property of *C* a value taken from its domain. A *e-schema* S over a template $(\mathsf{C}, \Sigma_{\mathsf{C}})$ is a finite set of e-schema components over constructs in C that satisfy Σ_{C}.

Example 5. A valid e-schema for the template of Example 4 is the following:

Relation

name
EMP
DEPT

Key

name	in
EmpName	EMP
DeptNo	DEPT

Attribute

name	nullable	in
Salary	true	EMP
Building	true	DEPT

FKey

name	in	refer
Dept	EMP	DEPT

It is easy to see that this e-schema represents a relational table `EMP` with `EmpName` as key, `Salary` as attribute and `Dept` as foreign key, and a relational table `DEPT` with `DeptNo` as key and `Building` as attribute.

Note that e-schemas in Example 5 is similar to the way in which commercial databases store metadata in catalogs. We can therefore easily verify whether a relational schema stored in a DBMS matches a given template definition: this can be done by querying the catalog of the system and checking the satisfaction of the dependencies.

In the following, an e-schema component over a construct $C(p_1, \ldots, p_k)$ will be called a *relation* component if $C = \mathsf{Relation}$, an *attribute* component if $C = \mathsf{Attribute}$, a *key* component if $C = \mathsf{Key}$, a *foreign key* component if $C = \mathsf{FKey}$. Moreover, we will denote an e-schema component over a construct $C(p_1, \ldots, p_k)$ by $C(p_1 : a_1, \ldots, p_k : a_k)$. Alternatively, we will use, for each construct, a tabular notation with a column for each property.

In the next sections, we describe how the notion of e-schema can be converted into a "standard" relational schema, and vice versa.

4.2. Relational decoding

Basically, in order to convert an e-schema into a relational schema we need to define the formulas describing the semantics of the various e-schema components, according to the intended meaning of the corresponding constructs.

Let S be an e-schema over a template $\mathscr{T} = (\mathsf{C}, \Sigma_{\mathsf{C}})$. The *relational decoding* of S is a pair $(\mathsf{S}, \Sigma_{\mathsf{S}})$ where:

- **S** contains a relation $R(K_1, \ldots, K_m, A_1, \ldots, A_n, F_1, \ldots, F_p)$ for each relation component $r \in \mathsf{S}$ such that:
  - $R = r(\mathsf{name})$,
  - $K_i = k_i(\mathsf{name})$ $(1 \leqslant i \leqslant m)$ for each key component $k_i \in \mathsf{S}$ such that $k_i(\mathsf{in}) = R$,
  - $A_j = a_j(\mathsf{name})$ $(0 \leqslant j \leqslant n)$ for each attribute component $a_j \in \mathsf{S}$ such that $a_j(\mathsf{in}) = R$,
  - $F_k = f_k(\mathsf{name})$ $(0 \leqslant k \leqslant p)$ for each foreign key component $f_k \in \mathsf{S}$ such that $f_k(\mathsf{in}) = R$.

- Σ_{S} contains an egd over the relation $R(K_1, \ldots, K_m, A_1, \ldots, A_n) \in \mathsf{S}$ of the form:

$$R(x_1, \ldots, x_m, y_1, \ldots, y_n), R(x_1, \ldots, x_m, y'_1, \ldots, y'_n) \rightarrow (y_1 = y'_1, \ldots, y_n = y'_n)$$

for each relation component $r \in \mathsf{S}$ such that:
- $R = r(\mathsf{name})$;
- $K_i = k_i(\mathsf{name})$ $(1 \leqslant i \leqslant m)$ for each key component $k_i \in \mathsf{S}$ such that $k_i(\mathsf{in}) = R$.

- Σ_{R} contains a tgd over a pair of relation schemas $R(A_1, \ldots, A_m, F_1, \ldots, F_n)$ and $R'(K_1, \ldots, K_n, A'_1, \ldots, A'_p)$ in **S** of the form:

$$R(x_1, \ldots, x_m, y_1, \ldots, y_n) \rightarrow R'(y_1, \ldots, y_n, z_1, \ldots, z_p)$$

for each pair of relation components r and r' in S such that:
- $R = r(\mathsf{name})$ and $R' = r'(\mathsf{name})$;
- $F_i = f_i(\mathsf{name})$ $(1 \leqslant i \leqslant n)$ for each foreign key component $f_i \in \mathsf{S}$ such that $f_i(\mathsf{in}) = R$ and $f_i(\mathsf{refer}) = R'$.

Example 6. Let us consider the e-schema S of Example 5, which is repeated here for convenience:

Relation

name
EMP
DEPT

Key

name	in
EmpName	EMP
DeptNo	DEPT

Attribute

name	nullable	in
Salary	true	EMP
Building	false	DEPT

FKey

name	in	refer
Dept	EMP	DEPT

The relational representation of S is the pair (S, Σ_S) where:

$$S = \{\texttt{EMP}(\texttt{EmpName}, \texttt{Salary}, \texttt{Dept}), \texttt{DEPT}(\texttt{DeptNo}, \texttt{Building})\}$$
$$\Sigma_S = \{\texttt{EMP}(x_1, x_2, x_3), \texttt{EMP}(x_1, x'_2, x'_3) \rightarrow (x_2 = x'_2, x_3 = x'_3), \texttt{DEPT}(x_1, x_2), \texttt{DEPT}(x_1, x'_2) \rightarrow (x_2 = x'_2), \texttt{EMP}(x_1, x_2, x_3) \rightarrow \texttt{DEPT}(x_3, x'_2)\}$$

In the same line, a procedure for the *encoding* of a relational schema, that is for the transformation of a relational schema (S, Σ_S) into an e-schema S, can also be defined. This procedure will be illustrated in the following section.

4.3. Relational encoding

Let **S** be a relational schema with a set of dependencies Σ_S including, as usual, egds and tgds. The *encoding* of **S** is an e-schema S such that:

- S contains a relation component r for each relation $R(A_1, \ldots, A_n) \in S$ such that $r(\text{name}) = R$;
- S contains an attribute component a for each attribute A of a relation $R \in S$ not involved in an egd or in a tgd Σ_S over R such that: $a(\text{name}) = A_i$ and $a(\text{in}) = R$;
- S contains a key component k_i $(1 \leqslant i \leqslant m)$ for each egd in Σ_S over a relation schema $R(K_1, \ldots, K_m, A_1, \ldots, A_n) \in S$ of the form:

$$R(x_1, \ldots, x_m, y_1, \ldots, y_n), R(x_1, \ldots, x_m, y'_1, \ldots, y'_n) \rightarrow (y_1 = y'_1, \ldots, y_n = y'_n)$$

such that: $k_i(\text{name}) = K_i$ and $k_i(\text{in}) = R$;
- S contains a foreign key component f_i $(1 \leqslant i \leqslant n)$ for each tgd over a pair of relation schemas $R(A_1, \ldots, A_m, F_1, \ldots, F_n)$ and $R'(K_1, \ldots, K_n, A'_1, \ldots, A'_p)$ in **S** of the form:

$$R(x_1, \ldots, x_m, y_1, \ldots, y_n) \rightarrow R'(y_1, \ldots, y_n, z_1, \ldots, z_p)$$

such that: $f_i(\text{name}) = F_i$, $f(\text{in}) = R$, and $f(\text{refer}) = R'$.

Example 7. Let us consider the relational schema (S, Σ_S) where:

$$S = \{\texttt{ORDER}(\texttt{OrderNo}, \texttt{Item}, \texttt{Quantity}), \texttt{PRODUCT}(\texttt{ProdNo}, \texttt{Price})\}$$
$$\Sigma_S = \{\texttt{ORDER}(x_1, x_2, x_3), \texttt{ORDER}(x_1, x'_2, x'_3) \rightarrow (x_2 = x'_2, x_3 = x'_3), \texttt{PRODUCT}(x_1, x_2), \texttt{PRODUCT}(x_1, x'_2) \rightarrow (x_2 = x'_2), \texttt{ORDER}(x_1, x_2, x_3) \rightarrow \texttt{PRODUCT}(x_2, x'_2)\}$$

The encoding of (S, Σ_S) is the following e-schema:

Relation

name
ORDER
PRODUCT

Key

name	in
OrderNo	ORDER
ProdNo	PRODUCT

Attribute

name	in
Quantity	ORDER
Price	PRODUCT

FKey

name	in	refer
Item	ORDER	PRODUCT

5. Schema exchange

In this section we define the schema exchange problem as the application of the data exchange problem to *templates* of schemas.

5.1. Source-to-target template dependency

In order to investigate the problem of data and metadata translation between templates we introduce the following notion.

Definition 3. s–t template dependency Given a *source* template $\mathscr{T}_1$ and a *target* template $\mathscr{T}_2$, a source-to-target (s–t) *template dependency* is a tuple generating dependency of the form: $\phi(x) \rightarrow \psi(x, y)$, where $\phi(x)$ is a conjunction of atomic formulas over the components of C_1 and $\psi(x, y)$ is a conjunction of atomic formulas over the components of C_2.

There are two different but equivalent semantics that can be associated with s–t template dependencies. In a declarative semantics, they represent constraints required to hold on pairs of instances over the source and the target template. Under this semantics, multiple pairs of e-schemas may satisfy the relationship. Alternatively, under the transformational semantics, the dependencies are interpreted as transformations from the source to the target: given a source e-schema S over the source template, there is a procedural way of computing a target e-schema S′ over the target template. This transformational semantics follows the data exchange line of work and it is the one we adopt in the schema exchange definition we introduce next.

5.2. The problem of schema exchange

Given a source template $\mathscr{T}_1 = (C_1, \Sigma_{C_1})$, a target template $\mathscr{T}_2 = (C_2, \Sigma_{C_2})$, and a set $\Sigma_{C_1C_2}$ of s–t template dependencies, we denote a *schema exchange setting* by the triple $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{C_1C_2})$.

Definition 4. Schema exchange Let $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{C_1C_2})$ be a schema exchange setting and S_1 a source e-schema over (C_1, Σ_{C_1}). The *schema exchange problem* consists in finding a finite target e-schema S_2 over (C_2, Σ_{C_2}) such that $S_1 \cup S_2$ satisfies $\Sigma_{C_1C_2}$. In this case S_2 is called *a solution for* S_1 or, simply, a *solution*.

Example 8. Consider a schema exchange problem in which the source template $\mathscr{T}_1 = (C_1, \Sigma_{C_1})$ and the target template $\mathscr{T}_2 = (C_2, \Sigma_{C_2})$ are the following:

$$C_1 = \{\mathsf{Relation}(\mathsf{name}), \mathsf{Key}(\mathsf{name}, \mathsf{in}), \mathsf{Attribute}(\mathsf{name}, \mathsf{in})\}$$
$$C_2 = \{\mathsf{Relation}(\mathsf{name}), \mathsf{Key}(\mathsf{name}, \mathsf{in}), \mathsf{Attribute}(\mathsf{name}, \mathsf{in}), \mathsf{FKey}(\mathsf{name}, \mathsf{in}, \mathsf{refer})\}$$

with the relational dependencies in Σ_{C_1} and in Σ_{C_2} shown in Example 4.

Assume now that we would like to split each relation over $\mathscr{T}_1$ into a pair of relations over $\mathscr{T}_2$ related by a foreign key and assign to the first relation in the target the name of the source relation. This scenario is graphically shown (informally) in Fig. 4 and is precisely captured by the following set of tgds Σ_{C_1,C_2}:

$$\Sigma_{C_1,C_2} = \{\mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{Attribute}(n_A, n_R) \rightarrow \mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{FKey}(n_F, n_R, n'_R), \mathsf{Relation}(n'_R), \mathsf{Key}(n_F, n'_R), \mathsf{Attribute}(n_A, n'_R)\}$$

We stress the fact that Fig. 4 just describes, in an easy-to-understand but informal way, a transformation that is instead precisely represented by the above dependency. Other representations could be used in a practical tool with a user friendly interface. This subject is however outside the goal of this paper which is rather devoted to the study of the fundamental properties of the schema exchange framework. We also point out that the construct names in the preconditions and in the conclusions have different meanings, since they refer to the source and the target respectively.

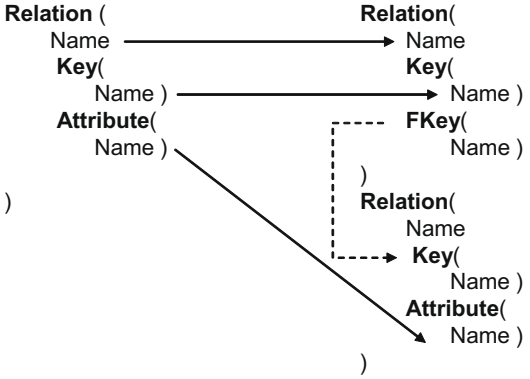

Fig. 4. Schema exchange setting for Example 8.

Consider now the following e-schema S for the template $\mathscr{T}_1$:

Relation

name
EMP

Key

name	in
EmpName	EMP

Attribute

name	in
DeptName	EMP
Floor	EMP

According to the decoding procedure described in Section 4.2, this e-schema encodes the relational schema: $S = \{\texttt{EMP}(\texttt{EmpName}, \texttt{DeptName}, \texttt{Floor})\}$ in which EmpName is the key. A possible solution S'_1 for this setting is:

Relation

name
EMP
R_1
R_2

Key

name	in
EmpName	EMP
K_1	R_1
K_2	R_2

Attribute

name	in
DeptName	R_1
Floor	R_2

FKey

name	in	refer
K_1	EMP	R_1
K_2	EMP	R_2

where R_1, R_2, K_1, K_2 are labelled nulls. The decoding of this solution contains three relations: $\texttt{EMP}(\underline{\texttt{EmpName}}, K_1, K_2)$, $R_1(\underline{K_1}, \texttt{DeptName})$, and $R_2(\underline{K_2}, \texttt{Floor})$, in which the attributes K_1, K_2 of relation EMP are foreign keys for R_1 and R_2, respectively. There are several null values because the dependencies in Σ_{C_1,C_2} do not allow the complete definition of the target e-schema.

Actually many other solutions for the same schema exchange problem exist. For instance the following e-schema S'_2:

Relation

name
EMP
R_1

Key

name	in
EmpName	EMP
K_1	R_1

Attribute

name	in
DeptName	R_1
Floor	R_1

FKey

name	in	refer
K_1	EMP	R_1

where R_1 and K_1 are labelled nulls. By decoding this solution we obtain two relations: $\texttt{EMP}(\underline{\texttt{EmpName}}, K_1)$ and $R_1(\underline{K_1}, \texttt{DeptName}, \texttt{Floor})$, where K_1 of relation EMP is a foreign key for R_1.

Two issues arise from Example 8: which solution to choose and how to generate it. Solution S'_2 in the example seems to be less general than S'_1. This is captured precisely by the notion of homomorphism. In fact, it is easy to see that, while there is a homomorphism from S'_1 to S'_2 ($R_2 \mapsto R_1, K_2 \mapsto K_1$), there is no homomorphism from S'_2 to S'_1. It follows that S'_2 contains "extra" information whereas S'_1 is a more general solution. As in data exchange [12,14], we argue that the "correct" solution is the most general one, in the sense above. This solution is called universal, as illustrated in the next section.

5.3. Universal solutions and core

The discussion on the possible solutions for a schema exchange problem leads to the following definition.

Definition 5. Universal solution A solution S of the schema exchange problem is *universal* if there exists a homomorphism from S to all the other solutions.

The next result provides a method for the generation of a universal solution of a given schema exchange problem. It follows from an analogous result for the data exchange problem.

Theorem 1. *Let $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{C_1C_2})$ be a data exchange setting and S_1 be an e-schema over $\mathscr{T}_1$ and assume that Σ_{C_2} includes only weakly acyclic tgds. The chase procedure over S_1 using $\Sigma_{C_1C_2} \cup \Sigma_{C_2}$ terminates and generates a universal solution if a solution exists and fails otherwise.*

Proof. A schema exchange setting can be viewed as a data exchange setting over a source schema S_1 and a target schema S_2 that involve a relation for each construct of $\mathscr{T}_1$ and $\mathscr{T}_2$, respectively. Under this view, S_1 is an instance of S_1. It follows that the results on data exchange shown in [14] can be applied to our scenario. In particular, the fact that, if a solution exists, the application of the chase procedure to an instance of S_1 using the given dependencies terminates and generates a universal solution. In this case, by construction, the output is an instance of $\mathscr{T}_2$ and, by definition, is also a universal solution of the given schema exchange problem. □

Theorem 1 provides a procedural way to generate a universal solution of a schema exchange problem. However, what we obtain is only one of the possible universal solutions and it turns out that, in general, it is not necessarily the best in terms of size. This is clarified in the following example.

Example 9. Consider a schema exchange problem in which the source template $\mathscr{T}_1 = (C_1, \Sigma_{C_1})$ and the target template $\mathscr{T}_2 = (C_2, \Sigma_{C_2})$ are the following:

$$C_1 = \{\mathsf{Relation}(\mathsf{name}), \mathsf{Key}(\mathsf{name}, \mathsf{in}), \mathsf{Attribute}(\mathsf{name}, \mathsf{in})\}$$
$$C_2 = \{\mathsf{Relation}(\mathsf{name}), \mathsf{Key}(\mathsf{name}, \mathsf{in})\}$$

with the usual relational constraints in Σ_{C_1} and in Σ_{C_2} shown in Example 4. Note that the second template models schemas with only key attributes.

Assume now that we would like just to copy each relation over $\mathscr{T}_1$ into a relation over $\mathscr{T}_2$ having a new key. This transformation can be implemented by the following tgd:

$$\Sigma_{C_1,C_2} = \{\mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{Attribute}(n_A, n_R) \rightarrow \mathsf{Relation}(n_R), \mathsf{Key}(n'_K, n_R)\}$$

Note that the rule is applicable only if the relation in the source contains at least one key and one attribute, even if the latter is not used in the target. Let us consider the following e-schema S over $\mathscr{T}_1$:

Relation

name
ORDER

Key

name	in
OrderNo	ORDER

Attribute

name	in
Item	ORDER
Quantity	ORDER

If we apply the chase over S using over Σ_{C_1,C_2} we obtain the following universal solution S′:

Relation

name
ORDER

Key

name	in
K_1	ORDER
K_2	ORDER

where K_1 and K_2 are labelled nulls that represent any key attribute names: they form together a composite key for the relation ORDER.

However, it is easy to see that the following e-schema S″ is also a universal solution:

Relation

name
ORDER

Key

name	in
K_1	ORDER

Note that S″ is homomorphically equivalent to S′ (that is, there is a homomorphism from S′ to S″ and vice versa) but it is more compact than S′.

Solutions S′ and S″ of Example 9 are both universal but S″ is clearly preferable to S′ since it is smaller in size. This solution is actually the *core* of all the universal solutions for S. It has been shown that the core is unique, as all universal solutions for a schema exchange problem have the same core up to isomorphism [12]. The core for a schema exchange problem with only egd as target constraints can be computed in naive way by successively removing tuples from a universal solution, as long as the solution resulting in each step satisfy the dependencies. Recently, Gottlob and Nash [16] have proposed a polynomial time algorithm that computes the core of a universal solution of a data exchange problem whose source-to-target constraints are tgds and whose target constraints consist of egds and weakly acyclic tgds. Hereinafter, we will only refer to the core of universal solutions.

6. From schema to data exchange

In this section we propose a transformation process that generates a *data* exchange setting as an instance of a schema exchange setting for a given data source.

6.1. Metalinks and S–D transformation process

Before discussing the transformation process, a preliminary notion is needed. In order to convert the schema exchange setting into a data exchange setting, we need to keep track of the correspondences between the source schema and the solution of the schema exchange problem. This can be seen as an application of the data provenance problem to schema exchange. To this end, by extending to our context a notion introduced in [11], we introduce the notion of *metalink* to describe the relationships between source and target metadata.

Definition 6. Metalink Let S be an e-schema and Σ be a set of s–t template dependencies. A *metalink* for S is an expression of the form $I \rightarrow_{\sigma,s} I\prime$ where $I \subseteq S$ and I' is the result of the application of a chase step on I based on the dependency $\sigma \in \Sigma$ and the substitution s.

Note that, since a reduced number of elements are involved in schema exchange, we can store all the metalinks and we do not need to compute them partially and incrementally as in [11].

Given a relational database over a schema S_1 and a schema exchange setting $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{C_1C_2})$ such that the encoding S_1 of S_1 is an instance of $\mathscr{T}_1$, we aim at generating a target database over a schema S_2 such that the encoding S_2 of S_2 is a universal solution for S_1. We call such process S–D *transformation* and it can be summarized as follows.

Input: A schema S_1 with constraint Σ_{S_1} and a schema exchange setting $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{C_1C_2})$;
Output: A data exchange setting $(S_1, S_2, \Sigma_{S_1S_2}, \Sigma_{S_2})$;
(1) Encode (S_1, Σ_{S_1}) into an e-schema S_1;
(2) Apply the chase procedure to S_1 using $\Sigma_{C_1C_2}$ and save the metalinks in a set $\mathscr{M}$ during the execution: each chase step based on the dependency $\sigma \in \Sigma$ with a substitution s adds to $\mathscr{M}$ a metalink $I \rightarrow_{\sigma,s} I'$;
(3) Decode the result S_2 of the chase procedure into a schema S_2 with constraints Σ_{S_2};
(4) For each metalink $I \rightarrow_{\sigma,s} I'$ in $\mathscr{M}$:
  (a) let L be the set of relations in S_1 mentioned in I: annotate all the attributes A in L with $s^{-1}(A)$;
  (b) let R be the set of relations in S_2 mentioned in I': annotate all the attributes A' in R with $s^{-1}(A')$;
  (c) replace all the attributes in L and R with fresh variables by associating the same variable to the attributes with the same annotation according to the constraints in Σ_{S_1} and Σ_{S_2};
  (d) add the tgd $L \rightarrow R$ to a set $\Sigma_{S_1S_2}$;
(5) Return the data exchange setting $(S_1, S_2, \Sigma_{S_1S_2}, \Sigma_{S_2})$.

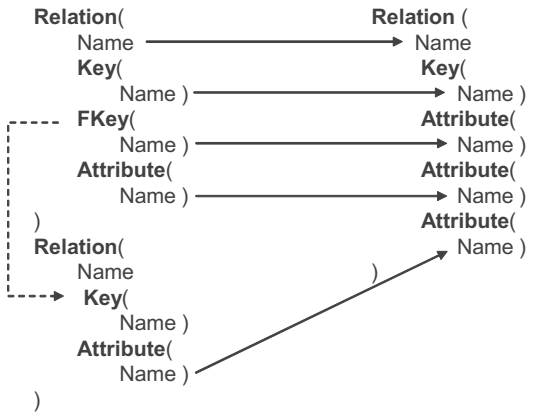

Fig. 5. Schema exchange scenario for Example 10.

Example 10. Let us consider the schema exchange setting described graphically in Fig. 5 and represented by the following set of tgds Σ_{C_1,C_2}:

$$\{v_1 = \mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{FKey}(n_F, n_R, n'_R), \mathsf{Attribute}(n_A, n_R), \mathsf{Relation}(n'_R), \mathsf{Key}(n'_K, n'_R), \mathsf{Attribute}(n'_A, n'_R) \rightarrow \mathsf{Relation}(n_R), \mathsf{Key}(n_K, n_R), \mathsf{Attribute}(n_F, n_R), \mathsf{Attribute}(n_A, n_R), \mathsf{Attribute}(n'_A, n_R)\}$$

Intuitively, the dependency occurring in Σ_{C_1,C_2} specifies that the target is obtained by joining two source relations according to a foreign key defined between them. Now consider the following source schema:

$$\mathsf{S} = \{\mathtt{EMP}(\mathtt{eid}, \mathtt{name}, \mathtt{did}), \mathtt{DEPT}(\mathtt{did}, \mathtt{dname})\},$$
$$\Sigma_{\mathsf{S}} = \{\mathtt{EMP}(x_1, x_2, x_3), \mathtt{EMP}(x_1, x'_2, x'_3) \rightarrow (x_2 = x'_2, x_3 = x'_3), \mathtt{DEPT}(x_1, x_2), \mathtt{DEPT}(x_1, x'_2) \rightarrow (x_2 = x'_2), \mathtt{EMP}(x_1, x_2, x_3) \rightarrow \mathtt{DEPT}(x_3, x'_1)\}$$

The encoding of **S** is the e-schema S that follows:

Relation

	name
s_1	EMP
s_2	DEPT

Key

	name	in
s_3	eid	EMP
s_4	did	DEPT

Attribute

	name	in
s_5	name	EMP
s_6	dname	DEPT

FKey

	name	in	refer
s_7	did	EMP	DEPT

Let $\{s_1, \ldots, s_7\}$ be the e-components of S. The application of the chase based on the given tgd produces the set of e-schema components $\{t_1, \ldots, t_5\}$:

Relation

	name
t_1	EMP

Key

	name	in
t_2	eid	EMP

Attribute

	name	in
t_3	name	EMP
t_4	did	EMP
t_5	dname	EMP

The metalink generated by this chase step is: $\{s_1,\ldots,s_7\}\rightarrow_{v_1,s_1}\{t_1,\ldots,t_5\}$, where s_1 is the substitution:

$$\{n_R\mapsto\texttt{EMP}, n_K\mapsto\texttt{eid}, n_F\mapsto\texttt{did}, n_A\mapsto\texttt{name}, n'_R\mapsto\texttt{DEPT}, n'_K\mapsto\texttt{did}, n'_A\mapsto\texttt{dname}\}$$

The chase ends successfully and produces an e-schema S′ whose decoding is the schema $(\mathsf{S}', \Sigma_{\mathsf{S}'})$ where:

$$\mathsf{S}' = \{\texttt{EMP}(\texttt{eid},\texttt{name},\texttt{did},\texttt{dname})\}$$
$$\Sigma_{\mathsf{S}'} = \{\texttt{EMP}(x_1,x_2,x_3,x_4), \texttt{EMP}(x_1,x'_2,x'_3,x'_4) \rightarrow (x_2 = x'_2, x_3 = x'_3, x_4 = x'_4)\}$$

Now, on the basis of the above metalink, we derive the following sets of annotated relations:

$$L = \{\texttt{EMP}(\texttt{eid}[n_K],\texttt{name}[n_A],\texttt{did}[n_F]), \texttt{DEPT}(\texttt{did}[n'_K],\texttt{dname}[n'_A])\}$$
$$R = \{\texttt{EMP}(\texttt{eid}[n_K],\texttt{name}[n_A],\texttt{did}[n_F],\texttt{dname}[n'_A])\}$$

By replacing all the attributes in L and R with variables in such a way that the attributes with the same annotation are replaced with the same variable, we obtain the following s–t tgd:

$$d_1 = \mathsf{S}.\texttt{EMP}(x_1,x_2,x_3), \mathsf{S}.\texttt{DEPT}(x_3,x_4) \rightarrow \mathsf{S}'.\texttt{EMP}(x_1,x_2,x_3,x_4)$$

The final data mapping scenario is reported graphically in Fig. 6.

As a final comment, we note that the example can be naturally extended to a more complex source schema. If, for instance, the relation EMP had two attributes, the tgd v_1 would fire twice, producing in the second chase step a further attribute schema component for EMP in the target e-schema, and therefore another attribute in the target schema.

6.2. Properties of the S–D transformation process

A number of general results on the S–D transformation process can be shown. First, the fact that the output of the process is a "correct" result, that is, the solution of the data exchange problem reflects the semantics of the schema exchange problem given as input. In order to introduce the concept of correctness in this context, a preliminary notion is needed.

Given an s–t tgd d over relational schemas, the *encoding* of d is a tgd over schema templates obtained by applying a procedure similar to the one defined in Section 4.3 for schemas.

More precisely, let d be an s–t tgd over a source schema **S** and a target schema **T**, and let Σ_{S} and Σ_{T} be two set of dependencies over **S** and **T**, respectively.

The *encoding* of d is an s–t template tgd $\hat{d}$ such that:

- the left-hand side (right-hand side) of the tgd $\hat{d}$ contains an atom of the form $\mathsf{Relation}(R)$ for each relation R occurring in the lhs (rhs) of d;
- the lhs (rhs) of $\hat{d}$ contains a set of atoms of the form $\mathsf{Key}(K_i, R)$ $(1 \leqslant i \leqslant m)$ for each relation $R(K_1,\ldots,K_m,A_1,\ldots,A_n)$ occurring in d such that there is an egd:

  $$R(x_1,\ldots,x_m,y_1,\ldots,y_n), R(x_1,\ldots,x_m,y'_1,\ldots,y'_n) \rightarrow (y_1 = y'_1,\ldots,y_n = y'_n)$$

  in Σ_{S} (Σ_{T});
- the lhs (rhs) of $\hat{d}$ contains a set of atoms of the form $\mathsf{FKey}(F_i, R, R')$ $(1 \leqslant i \leqslant n)$ for each relation $R(A_1,\ldots,A_m,F_1,\ldots,F_n)$ occurring in d such that there is a tgd:

  $$R(x_1,\ldots,x_m,y_1,\ldots,y_n) \rightarrow R'(y_1,\ldots,y_n,z_1,\ldots,z_p)$$

  in Σ_{S} (Σ_{T});
- the lhs (rhs) of $\hat{d}$ contains a set of atoms of the form $\mathsf{Attribute}(A_i, R)$ $(1 \leqslant i \leqslant p \leqslant n)$ for each relation $R(K_1,\ldots,K_m,A_1,\ldots,A_n)$ occurring in the lhs (rhs) of d such that A_i in not involved in the rhs of an egd in Σ_{S} (Σ_{T}) or in both the lhs and rhs of a tgd in Σ_{S} (Σ_{T}).

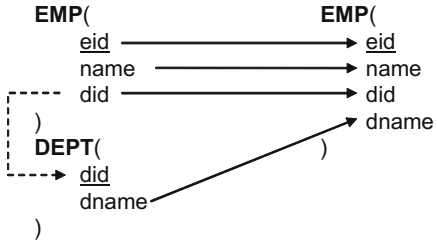

Fig. 6. Data exchange scenario for Example 10.

Example 11. Let us consider the tgd d_1 of Example 10, which is reported here for convenience:

$$d_1 = \mathsf{S}.\mathtt{EMP}(x_1, x_2, x_3), \mathsf{S}.\mathtt{DEPT}(x_3, x_4) \rightarrow \mathsf{S}'.\mathtt{EMP}(x_1, x_2, x_3, x_4)$$

The encoding of d_1 is the following s–t template tgd.

$$\begin{aligned} v_2 = {} & \mathsf{Relation}(\mathtt{EMP}), \mathsf{Key}(\mathtt{eid}, \mathtt{EMP}), \mathsf{Attribute}(\mathtt{name}, \mathtt{EMP}), \mathsf{FKey}(\mathtt{did}, \mathtt{EMP}, \mathtt{DEPT}), \\ & \mathsf{Relation}(\mathtt{DEPT}), \mathsf{Key}(\mathtt{did}, \mathtt{DEPT}), \mathsf{Attribute}(\mathtt{dname}, \mathtt{DEPT}) \rightarrow \mathsf{Relation}(\mathtt{EMP}), \mathsf{Key}(\mathtt{eid}, \mathtt{EMP}), \\ & \mathsf{Attribute}(\mathtt{name}, \mathtt{EMP}), \mathsf{Attribute}(\mathtt{did}, \mathtt{EMP}), \mathsf{Attribute}(\mathtt{dname}, \mathtt{EMP}) \end{aligned}$$

The tgd v_2 in the example above is less general than the original tgd v_1 for the schema exchange scenario described in Example 10. However, it generates the same output S′ on the given input S. This exactly captures the fact that the data exchange problem obtained as output captures the semantics of the schema exchange problem given as input.

This intuition is captured by the following correctness result.

Theorem 2. *Let* $(\mathsf{S}, \mathsf{S}', \Sigma_{\mathsf{SS}'})$ *be the output of the* S–D *transformation process when* $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{\mathsf{C}_1\mathsf{C}_2})$ *and* **S** *are given as input and let* $\widehat{\Sigma}$ *be the set of s–t tgds obtained by encoding the s–t tgds in* $\Sigma_{\mathsf{SS}'}$. *The encoding* S′ *of* **S**′ *is a universal solution for the encoding* S *of* **S** *under the schema exchange setting* $(\mathscr{T}_1, \mathscr{T}_2, \widehat{\Sigma})$.

Proof. The e-schema S′ is obtained in step 2 of the S–D transformation process by chasing the encoding S of **S** using the dependencies in $\Sigma_{\mathsf{C}_1\mathsf{C}_2}$. The proof proceeds by induction on the number n of chase steps needed to generate S′. Specifically, we show that, for every $0 \leqslant i \leqslant n$, indicating with S^i the e-schema produced after the ith step in chasing S using dependencies in $\Sigma_{\mathsf{C}_1\mathsf{C}_2}$, there is an e-schema $\widehat{\mathsf{S}^i}$ produced as an intermediate result in chasing S using dependencies in $\widehat{\Sigma}$ such that S^i is homomorphic to $\widehat{\mathsf{S}^i}$. Since, by Theorem 1, if a solution exists, $\widehat{\mathsf{S}^n}$ is a universal solution for S under $(\mathscr{T}_1, \mathscr{T}_2, \widehat{\Sigma})$, if there is a homomorphism from $\mathsf{S}^n = \mathsf{S}\prime$ to $\widehat{\mathsf{S}^n}$ then the former is also a universal solution for S under $(\mathscr{T}_1, \mathscr{T}_2, \widehat{\Sigma})$ and the claim follows. The basis is immediate since, for $n = 0$, the e-schema S′ as well as the sets of dependencies $\Sigma_{\mathsf{SS}'}$ and $\widehat{\Sigma}$ are empty by construction, and so S′ is also a trivial universal solution for $(\mathscr{T}_1, \mathscr{T}_2, \widehat{\Sigma})$. With respect to the induction, assume that there is a homomorphism h^i from S^{i-1} to $\widehat{\mathsf{S}^{i-1}}$ and let $\mathsf{S}^{i-1} \rightarrow_{\sigma,s} \mathsf{S}^i$ be the metalink corresponding to the ith chase step over S based on the tgd $\sigma \in \Sigma_{\mathsf{C}_1\mathsf{C}_2}$ and the substitution s. The result of the application of this chase step is, by definition of chase step, $\mathsf{S}^i = \mathsf{S}^{i-1} \cup s(R)$, where R is the right-hand side of σ. By construction (step 4 of the S–D transformation process), there is a tgd $\sigma' \in \Sigma_{\mathsf{SS}'}$ which is built from the above metalink by associating the same variable to the attributes of **S** and **S**′ that are mapped by σ. Since the encoding of a tgd preserves the bindings of the variables, it follows that there is a substitution s^* from σ to $\hat{\sigma}$. Moreover, by definition of encoding of a tgd it easily follows that there is a substitution $\hat{s}$ from the left-hand side $\widehat{L}$ of $\hat{\sigma}$ to the encoding $\underline{\mathsf{S}}$ of **S**. Since S is not modified by the chase procedure, we have that $\hat{s}$ is also a substitution from the left-hand side of$\hat{\sigma}$ to S^{i-1}. Hence, we can apply a chase step based on $\hat{\sigma}$ and $\hat{s}$ to $\widehat{\mathsf{S}^{i-1}}$. The result of the application of this chase step is, by definition, $\widehat{\mathsf{S}^i} = \widehat{\mathsf{S}^{i-1}} \cup \hat{s}(\widehat{R})$, where $\widehat{R}$ is the right-hand side of $\hat{\sigma}$, and since $\widehat{R} = s^*(R)$, we have that $\widehat{\mathsf{S}^i} = \widehat{\mathsf{S}^{i-1}} \cup \hat{s}(s^*(R))$. Now let L be the left-hand side of σ: since, by the inductive hypothesis, there is a homomorphism h^i from S^{i-1} and S^{i-1}, we have that $h^i \circ s$ is a homomorphism from L to $\widehat{\mathsf{S}^{i-1}}$. At the same time, the composition of the substitutions s^*, from L to $\widehat{L}$, and $\hat{s}$, from $\widehat{L}$ to $\widehat{\mathsf{S}^{i-1}}$, is also a substitution from L to $\widehat{\mathsf{S}^{i-1}}$. It follows that, on the variables occurring in L and R, $h^i \circ s = \hat{s} \circ s^*$. Hence, we have that $h^i(\mathsf{S}^i) = h^i(\mathsf{S}^{i-1}) \cup h^i(s(R)) = \widehat{\mathsf{S}^{i-1}} \cup \hat{s}(s^*(R)) = \widehat{\mathsf{S}^i}$, and so h^i is a homomorphism from S^i to $\widehat{\mathsf{S}^i}$. □

The following completeness result can also be shown. We say that a data exchange setting is *constant-free* if no constants are used in formulas.

Theorem 3. *Any constant-free data exchange setting can be obtained from the* S–D *transformation process over some schema exchange setting.*

Proof. Given a data exchange setting $M = (\mathsf{S}, \mathsf{S}', \Sigma_{\mathsf{SS}'})$ we can derive a schema exchange setting $(\mathscr{T}_1, \mathscr{T}_2, \Sigma_{\mathsf{C}_1\mathsf{C}_2})$ from which M can be obtained with the S–D transformation process by: (i) defining two templates $\mathscr{T}_1$ and $\mathscr{T}_2$ such that **S** is an instance of $\mathscr{T}_1$ and **S**′ is an instance of $\mathscr{T}_2$ (this can be easily done on the basis of the structure of the encodings of **S** and **S**′), and (ii) encoding the tgds in $\Sigma_{\mathsf{SS}'}$. The schema exchange setting we obtain is already suitable for our purposes, but it can be made even more general by replacing each constant with a unique variable. It is also easy to see that this cannot be done if constants appear in $\Sigma_{\mathsf{SS}'}$ since the encoding procedure does not consider them. □

Intuitively, the above theorem states that a schema exchange can be generalized by substituting distinct variables for constants and since the S–D transformation does not handle constants the data exchange setting must be constant-free.

7. Related work

To our knowledge, the notion of schema exchange studied in this paper is new. In general, we can say that our contribution can be set in the framework of *metadata management*. Metadata can generally be thought as information that describes, or supplements, actual data. Several studies have addressed metadata related problems, such as, interoperability [18,23,29],

annotations and comments on data [7,10,15], data provenance [9], and a large list of more specific problems, like data quality [20]. While the list is not exhaustive, it witnesses the large interest in this important area and the different facets of the problem.

Most of the proposed approaches focus on a specific kind of metadata and are not directly applicable to other cases without major modifications. Bernstein set the various problems within a very general framework called *model management* [3–5]. In [6] the authors show the value of this framework to approach several metadata related problems, with a significant reduction of programming effort. Our contribution goes in this direction: as in model management, schemas and mappings are treated as first class citizens.

In particular, the schema exchange problem offers some novel research opportunity in the context of the *ModelGen* operator. The ModelGen operator realizes a *schema translation* from a source data model M_s to a target data model M_t and returns an executable mapping (or a transformational script) for the translation of the instances of the source schema. For instance, the ModelGen operator could be used to translate a relational database into a schema for an XML document (e.g., a DTD). ModelGen has been implemented in commercial products as a non-generic way to translate schemas between specific pairs of data models. In particular, the most supported scenario in available tools is the translation of ER diagrams into relational schemas.

Several proposals for the general problem have also been proposed in the last years [1,21,25]. All the frameworks share the same approach: (i) the system rewrites a source schema S into a representation S' for a universal metamodel; (ii) a sequence of rule-based transformations modifies S' to translate or eliminate the constructs that are not allowed in the target data model; (iii) after n transformations, S' can be rewritten into the representation for the target data model. It is evident that in this translation process a crucial role is done by the rule-based transformations. Surprisingly all the frameworks lack a tool for the design of such transformations at the data model level. In this paper, we provide a novel contribution to this problem by studying a framework for schema translation with a clear and precise semantics, that can be at the basis of an innovative tool supporting the design and the automatic generation of transformations over schemas.

It is also important to locate schema exchange in the context of data exchange. Our work is largely inspired by the theoretical foundations explored by Fagin et al. in the last years [12,14]. We remark that schema exchange is the first proposal to extend their results to metadata and to introduce the novel notion of encoding of schemas and tgds. One of the most important practical contributions in data exchange are the algorithms for the generation of the tgds representing the schema mappings [13,27]. Those algorithms take as input the schemas (with their constraints) and the arrows between the elements of the schemas (named *correspondences* in literature). We point out that in many practical settings the target schema does not exist, and it must be designed from scratch with a manual process before designing the mapping. This means that the user must deal with at least two manual steps: (i) the definition of the target schema and (ii) the definition of the correspondences between the elements. Many attempts have been done to automate the generation of the target schema [1,8,19,21,24,25] and the identification of the correspondences between different schemas (the *schema matching* problem, see [30] for a survey). Proposed solutions gave important but partial contributions: in general settings, with existing tools it is not possible to avoid the manual time-consuming work we have highlighted. Our approach effectively tackles this problem. As we have shown, in many cases it is possible to define generic transformations over templates of schemas and the use of these transformations can support the activity described above. Finally, we mention that the notion of template has been used in many contexts to support the design and reuse of software components [28].

Some of the results of this paper have been presented, in a preliminary form, in [26]. In this paper we have extended our earlier work in several ways: we have added a lot of practical examples to make more clear the various notions and results; we have added a new section (Section 2) that illustrates a motivating scenario and provides an overview of the approach; we have refined the notions of schema exchange, encoding and decoding of e-schemas, s–t dependencies and metaroute (now called metalink); we have introduced a new and more general S–D transformation process (now the new algorithm is self-contained as it does not refer to notions and techniques defined elsewhere); we have included the proofs for all the results; we have added a new section (Section 7) discussing related works.

8. Conclusion and future work

In this paper, we have introduced and studied the schema exchange problem, a generalization of data exchange to sets of schemas with similar structure. This problem consists of taking a schema that matches a source template and generating a new schema for a target template, on the basis of a set of dependencies defined over the two templates. To tackle this problem, we have presented a method for the generation of a "correct" solution of the problem and a process aimed at automatically generating a data exchange setting from a schema exchange solution.

As a follow-up to this theoretical study, we have developed a prototype, called GAIA, that relies on the theoretical framework illustrated in this paper and supports several activities involved in the management of heterogeneous data sources; in particular, the design and reuse of schema mappings. Specifically, GAIA provides a graphical interface that allows the user to: (i) describe data collections presenting structural similarities, by means of a source template T_1, (ii) define the structure of a possible transformation of the source by means of a target template T_2 and a set correspondences between T_1 and T_2, graphically represented by lines, (iii) translate any data source over a schema matching with T_1 into a format described

by a schema matching with T_2, (iv) make use of a set of predefined template mappings as *design patterns* for the development of new schema mappings, and (v) convert a schema mapping into a template mapping to be reused in a different application. GAIA asks user intervention when, for instance, there is the need to choose appropriate names for nulls occurring in schemas. We have successfully tested this tool in a number of cases from industrial settings. Further information on this tool can be found in: http://gaia.dia.uniroma3.it/.

We believe that other interesting directions of research can be pursued within the schema exchange settings. We sketch some of them:

- *Combining data and metadata*. The framework we have presented can be extended to support mappings and constraints involving data and metadata at the same time [17]. This scenario allows the user to specify the transformation of metadata into data and vice versa. For instance, we could move the name of a relational attribute into a tuple of a relation, or we could generate a target schema with a set of attributes identified by some data in the source. It is evident that, adding support to data and metadata in the same dependency, is a required step to extend the class of data exchange settings that can be generated in the S–D transformation process.
- *Reuse of existing data exchange*. We have developed a preliminary technique for generalizing data exchange mappings given by the user into a generic mapping over templates. The idea is to extract the design pattern for each data exchange scenario, and possibly find common patterns when the given scenarios are two or more. Once a transformation is expressed in a schema exchange setting, it can be later used to derive a data exchange transformation similar to the original one for a different pair of schemas.
- *Metaquerying*. A template is actually a schema and it can therefore be queried. A query over a template is indeed a *meta*-query since it operates over metadata. There are a number of metaqueries that are meaningful. For instance, we can retrieve with a query over a template the pairs of relations that can be joined, being related by a foreign key. Also, we can verify whether there is a join path between two relations.
- *Special class of solutions*. Given a schema exchange problem, can we verify whether all the solutions of the problem satisfy some relevant property? For instance, we would like to obtain only relations that are acyclic or satisfy some normal form. We are also investigating under which conditions a schema exchange problem generates a data exchange setting with certain properties, e.g., the fact that the dependencies belong to some relevant classes.

References

[1] P. Atzeni, P. Cappellari, P.A. Bernstein, Model-independent schema and data translation, in: International Conference on Extending Database Technology (EDBT), 2006, pp. 368–385.
[2] C. Beeri, M.Y. Vardi, A proof procedure for data dependencies, Journal of ACM 31 (4) (1984) 718–741.
[3] P.A. Bernstein, Applying model management to classical meta data problems, in: Conference on Innovative Data Systems Research (CIDR), 2003, pp. 209–220.
[4] P.A. Bernstein, S. Melnik, Model Management 2.0: Manipulating Richer Mappings, in: SIGMOD Conference, 2007, pp. 1–12.
[5] P.A. Bernstein, A.Y. Levy, R.A. Pottinger, A vision for management of complex models, SIGMOD Record 29 (4) (2000) 55–63.
[6] P.A. Bernstein, E. Rahm, Data warehouse scenarios for model management, in: International Conference on Conceptual Modeling (ER), 2000, pp. 1–15.
[7] D. Bhagwat, L. Chiticariu, W.C. Tan, G. Vijayvargiya, An annotation management system for relational databases, in: International Conference on Very Large Data Bases (VLDB), 2004, pp. 900–911.
[8] S. Bowers, L.M.L. Delcambre, The uni-level description: a uniform framework for representing information in multiple data models, in: International Conference on Conceptual Modeling (ER), 2003, pp. 45–58.
[9] P. Buneman, S. Khanna, W.C. Tan, Why and where: a characterization of data provenance, in: International Conference on Database Theory (ICDT), 2001, pp. 316–330.
[10] P. Buneman, S. Khanna, W.C. Tan, On propagation of deletion and annotations through views, in: Symposium on Principles of Database Systems (PODS), 2002, pp. 150–158.
[11] L. Chiticariu, W.C. Tan, Debugging schema mappings with routes, in: International Conference on Very Large Data Bases (VLDB), 2006, pp. 79–90.
[12] R. Fagin, P.G. Kolaitis, L. Popa, Data exchange: getting to the core, ACM Transaction on Database Systems 30 (1) (2005) 174–210.
[13] A. Fuxman, M.A. Hernández, H. Ho, R.J. Miller, P. Papotti, L. Popa, Nested mappings: schema mapping reloaded, in: International Conference on Very Large Data Bases (VLDB), 2006, pp. 67–78.
[14] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: semantics and query answering, Theoretical Computer Science 336 (1) (2005) 89–124.
[15] F. Geerts, A. Kementsietsidis, D. Milano, MONDRIAN: annotating and querying databases through colors and blocks, in: International Conference on Data Engineering (ICDE), 2006, pp. 82–93.
[16] G.Gottlob, A. Nash, Data exchange: computing cores in polynomial time, in: Symposium on Principles of Database Systems (PODS), 2006, pp. 40–49.
[17] M.A. Hernández, P. Papotti, W.C. Tan, Data exchange with data–metadata translations, Proceedings of the VLDB Endowment 1 (1) (2008) 260–273.
[18] L.V.S. Lakshmanan, F. Sadri, S.N. Subramanian, SchemaSQL: an extension to SQL for multidatabase interoperability, ACM Transaction on Database Systems 26 (4) (2001) 476–519.
[19] P. McBrien, A. Poulovassilis, Data integration by bi-directional schema transformation rules, in: International Conference on Data Engineering (ICDE), 2003, pp. 227–238.
[20] G. Mihaila, L. Raschid, M.-E. Vidal, Querying "quality of data" metadata, in: IEEE Meta-Data Conference, 1999.
[21] P. Mork, P.A. Bernstein, S. Melnik, Teaching a schema translator to produce O/R views, in: International Conference on Conceptual Modeling (ER), 2007, pp. 102–119.
[22] OMG Model Driven Architecture, Internet document, 2001. <http://www.omg.org/mda/>.
[23] OASIS XRI Data Interchange (XDI), Internet document, 2008. <http://www.oasis-open.org/>.
[24] P. Papotti, R. Torlone, Heterogeneous data translation through XML conversion, Journal of Web Engineering 4 (3) (2005) 189–204.
[25] P. Papotti, R. Torlone, Automatic generation of model translations, in: International Conference on Advanced Information Systems Engineering (CAiSE), 2007, pp. 36–50.
[26] P. Papotti, R. Torlone, Schema exchange: a template-based approach to data and metadata translation, in: International Conference on Conceptual Modeling (ER), 2007, pp. 323–337.

[27] L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernández, R. Fagin, Translating web data, in: International Conference on Very Large Data Bases (VLDB), 2002, pp. 598–609.
[28] S. Purao, V. Storey, A. Sengupta, M. Moore, Reconciling and cleansing: an approach to inducing domain models, in: International Workshop on Information Systems and Technologies (WITS), 2000, pp. 61–66.
[29] C.M. Wyss, E. Robertson, Relational languages for metadata integration, ACM Transaction on Database Systems 30 (2) (2005) 624–660.
[30] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB Journal 10 (4) (2001) 334–350.

Paolo Papotti is currently a Post Doctoral Researcher at the Department of Computer Engineering and Automation at Università Roma Tre. He has been a visiting scientist at the IBM Almaden research center (CA, USA) and a Post Doc. Scholar at the University of California Santa Cruz (USA).

His research interests are in data integration and data exchange. In particular, he is interested in techniques that improve and make automatic the process of searching and organizing data. His work has been presented in workshops, journals, and all the major international conferences of the field, including ACM-Sigmod, VLDB and IEEE-ICDE. He has submitted two invention disclosures at the US Patent of Technology and he has served as reviewer for international conferences, workshops and journals.

Riccardo Torlone is a full professor in the area of Information Systems at Università Roma Tre. He received his Dr. Ing. degree in Electrical Engineering from Università di Roma "La Sapienza". Before joining Università Roma Tre, he was member of the research staff at IASI-CNR in Rome, where he has still a research appointment. He also had a visiting research appointment at the University of California Los Angeles.

His research has considered various topics in the database field, including: relational database theory, active and deductive databases, tools for database design, models and languages for object-oriented databases, data warehouses and OLAP systems, Web based information systems, integration and transformation of heterogeneous data sources, adaptive systems and personalization. He has published his research results in the major journals of the field and in the refereed proceedings of all the major conferences.

He has authored the most spread book on databases in Italy, published also in an international edition and in several versions, and has authored two other books. He has organized many national and international scientific events (conferences and schools) and is currently serving as a member of the EDBT Association. He is member of the editorial board of an international journal on computer science and has been program committee member for many international conferences. He is currently the chair of the graduate and undergraduate programs in Computer Engineering at Università Roma Tre.